University
of Southern
California

A Gentle Introduction to Soar, an Architecture for Human Cognition

May 1996

ISI/RS-96-439

Jill Fain Lehman
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

John E. Laird
Dept. of EECS
University of Michigan
Ann Arbor, MI 48109

Paul S. Rosenbloom
USC/Information
Sciences Institute
Marina del Rey, CA 90292

INFORMATION
SCIENCES
INSTITUTE

310/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

A Gentle Introduction to Soar, an Architecture for Human Cognition

May 1996

ISI/RS-96-439

Jill Fain Lehman
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

John E. Laird
Dept. of EECS
University of Michigan
Ann Arbor, MI 48109

Paul S. Rosenbloom
USC/Information
Sciences Institute
Marina del Rey, CA 90292

This paper also appears in S. Sternberg & D. Scarborough (eds.), *An Invitation to Cognitive Science, Part IV: Conceptual and Methodological Foundations*, MIT Press, Cambridge, MA, 1995. In press.

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1996	Research Report

4. TITLE AND SUBTITLE

A Gentle Introduction to Soar, an Architecture for Human Cognition

5. FUNDING NUMBERS

None

6. AUTHOR(S)

Jill Fain Lehman, John Laird, and Paul Rosenbloom

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

ISI/RS-96-439

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

None listed

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

This paper also appears in S. Sternberg & D. Scarborough (eds.), An Invitation to Cognitive Science, Part IV: Conceptual and Methodological Foundations, MIT Press, Cambridge, MA, 1995. In press.

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

None

14. SUBJECT TERMS

cognitive architecture, cognitive science, Soar

15. NUMBER OF PAGES

46

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Table of Contents

1. Introduction 1
2. The idea of architecture 3
3. What cognitive behaviors have in common 5
4. Behavior as movement through problem spaces 9
5. Tying the content to the architecture 13
6. Memory, perception, action, and cognition 16
7. Detecting a lack of knowledge 23
8. Learning 26
9. Putting it all together: a Soar model of Joe Rookie 31
10. Stepping back: the Soar architecture in review 34
11. From architecture to unified theories of cognition 36
12. References 38

List of Figures

Figure 2-1: Levels of processing. The middle level can be viewed as both architecture for the level above and content for the level below. As an architecture WordPerfect™ offers a fixed set of methods for composing lots of different documents; as content, it is just one of many word-processing programs you could run on a Macintosh. 4

Figure 3-1: Joe's curve ball is hit by the batter, caught by Joe on a single hop and thrown to first base. 8

Figure 4-1: Looking at Joe's behavior as a space of decisions made over time. From the initial situation of holding the ball, Joe must decide what pitch to throw. On the basis of the outcome, he will make his next choice, and so on. 10

Figure 4-2: Behavior through time represented as movement through a problem space. The problem space is represented by a triangle to symbolize the ever-expanding set of possibilities that could unfold over time. The goal is represented by a circle at the apex of the triangle. Squares represent states, the features and values that reflect the internal and external situation. Goal states, states in which features have values that indicate the goal has been achieved, are shaded. Arrows represent operators that change or transform states. A state transformation may correspond to internal behavior (conscious or unconscious) or external behavior (actions observable in the world), or both. The states, features (bold face), and values (italics) are identified with arbitrary symbols, e.g. S1, f1 *v2*, for convenience. 12

Figure 5-1: A goal context for Joe. 14

Figure 6-1: Selecting and throwing the curve ball. In the first wave of the elaboration phase, the context in (a) triggers a3, adding information about the batter's handedness to the state. This change then triggers associations a4 and a5 in the second wave, producing proposals for throw-curve and throw-fast-ball. The operator proposals trigger a6 in the third wave, producing a preference for throwing the curve ball. No more associations fire so Soar enters the decision phase (c). The result of the decision procedure is to select the throw-curve operator for the operator slot of the goal context. The changes to the state that result from applying this operator actually occur during the elaboration phase of the next decision cycle (d). 20

Figure 7-1: (a) An operator-tie impasse in Pitch causes a new context to arise. When the decision cycle can't decide between the two proposed operators, the architecture creates an impasse structure (crescent) which links the current goal (get-batter-out) to its subgoal (resolve-tie), producing a goal-subgoal hierarchy. (b) Knowledge in the domain content then determines the rest of the subgoal context, i.e. the problem 24

space to work in to achieve the subgoal, the initial state and the operators that are applicable to it. The decision cycle may now result in a single change to either goal context in the hierarchy. In general, processing will continue in the lower problem space (Recall). If the decision cycle suggests a change to a slot in the higher goal context (Pitch), however, that change takes precedence and the lower context will disappear.

Figure 8-1: Learning when to prefer the curve ball. Elements in the higher goal context (the pre-impasse environment (a)) are used by operators available in the lower goal context (b) to elicit knowledge that resolves the operator tie. Here, the model elicits a "memory" of a previous game (event116) in which it was windy and throwing the fast ball to a left-handed batter led to a home run. On this basis, throwing the curve becomes preferred. The new association that will result from the resolution of this impasse will test for only those elements in the pre-impasse environment that led to the result, i.e. the windy day, left-handedness and the proposals of the two pitches. 28

Figure 9-1: Expanding the model's capabilities. Our model of Joe needs far more knowledge to play out our original scenario than we have given it in the previous sections. Still, increasing the model's functionality can be done within the architectural framework we've described by adding more domain content in the form of goal contexts and LTM associations that map between them. 32

List of Tables

Table 3-1: A small portion of the knowledge needed to model Joe Rookie. 9

Table 6-1: Some of the associations in the model's long-term memory. In the decision cycle, patterns of percepts and state features and values in WM are matched to the "if" portion of an association in LTM, and changes to the WM state and motor actions are produced by the associated "then" portion. 18

A Gentle Introduction to Soar, an Architecture for Human Cognition

Jill Fain Lehman, John Laird, Paul Rosenbloom

1. Introduction

Many intellectual disciplines contribute to the field of cognitive science: psychology, linguistics, anthropology, and artificial intelligence, to name just a few. Cognitive science itself originated in the desire to integrate expertise in these traditionally separate disciplines in order to advance our insight into cognitive phenomena — phenomena like problem solving, decision making, language, memory, and learning. Each discipline has a history of asking certain types of questions and accepting certain types of answers. And that, according to Allen Newell, a founder of the field of artificial intelligence, is both an advantage and a problem.

The advantage of having individual disciplines contribute to a study of cognition is that each provides expertise concerning the questions that have been at the focus of its inquiry. Expertise comes in two packages: descriptions of regularities in behavior, and theories that try to explain those regularities. For example, the field of psycholinguistics has been responsible for documenting a regularity called the *garden path phenomenon* which contrasts sentences like (a) that are very easy for people to understand, with sentences like (b), that are so difficult that most people believe they are ungrammatical (they're not, but their structure leads people to misinterpret them):

(a) Without her contributions we failed.
(b) Without her contributions failed to come in.

In addition to providing examples of garden path sentences and experimental evidence demonstrating that people find them nearly impossible to understand, psycholinguists have also constructed theories of why they are problematic, (e.g. (Gibson, 1990; Pritchett, 1988)). Indeed, psycholinguists are interested in phenomena like these because they help to constrain theories of how humans understand language; such a theory should predict that people will have problems on sentences like (b) but not on sentences like (a).

Psychology has also given us descriptions and theories of robust regularities, i.e. behaviors that all people seem to exhibit. It has contributed regularities about motor behavior (e.g. Fitts' Law (Fitts, 1954), which predicts how long it will take a person to move a pointer from one place to a

target location as a function of the distance to be traveled and the size of the target), about item recognition (e.g. that the time to decide whether a test item was on a memorized list of items increases linearly with the length of the list (Sternberg, 1975)), and about verbal learning (e.g. if an ordered list of items is memorized by repeated exposure, then the items at the ends of the list are learned before the items in the middle of the list (Tulving, 1983)), among others. The other disciplines mentioned above have also described and theorized about regularities in human cognition. How can the expert contributions of all these fields be a problem?

The problem arises not because of the regularities, but because of the theories. Each individual discipline really contributes what Newell called *microtheories* — small pieces of the big picture developed without the constraint of having to fit in with all the other pieces. It is true that the structures and mechanisms that underlie any particular regularity need not underlie every other regularity. But they must at least be compatible with the structures and mechanisms underlying those other regularities; after all, the regularities we observe in human cognition are all produced by a single system, the mind. If we think about cognition as a big picture, then a microtheory is a way of cutting a portion of that picture into a jigsaw puzzle. Each theory may cut up its own portion, even a portion that overlaps with another theory's, in a different way. So, when each discipline throws its set of pieces out on the table, how do we know that there is any set of pieces that will allow us to recover the big picture? The only way, Newell argues, is to go ahead and try to put the whole picture together, to try to build *unified theories of cognition* (UTCs) (Newell, 1990). So, around 1980, Newell and two of his students (the second and third authors of this chapter) began working on Soar, a candidate UTC.

As we said above, each of the disciplines that contributes to cognitive science has a history of asking certain types of questions and accepting certain kinds of answers. This is just another way of saying disciplines differ as to what constitutes a theory. For some, a theory is a set of equations, for others it is a computer program, for still others it is a written narrative that makes reference to some objects and ideas that are new in the context of other objects and ideas that are generally accepted. Although the form may differ, the purpose a theory serves — to answer questions — does not. Like Newell, our backgrounds are in both artificial intelligence and psychology. For that reason, the form of theory we are particularly interested in is mechanistic, that is, a theory as an account of underlying mechanisms and structures. Moreover, because artificial intelligence has its roots in computer science, we prefer to state our theory both

narratively and as a working computer program. So, for us at least, working on a unified theory of cognition means trying to find a set of computationally-realizable mechanisms and structures that can answer all the questions we might want to ask about cognitive behavior. A key piece of the puzzle, we believe, lies in the idea of architecture.

2. The idea of architecture

The idea of architecture is not new. In working with computers, we often describe and compare hardware architectures: the set of choices the manufacturer makes for a particular computer's memory size, commands, processor chip, etc. As a trip to any computer store will demonstrate, many hardware configurations are possible. Differences among hardware architectures reflect, in part, designs that are intended to be optimal under different assumptions about the software that architecture will process. Thus, once the many decisions about the hardware components and how they interact are made, the resulting architecture can be evaluated (or compared to that of another machine) only by considering how well it processes software. In other words, asking "Does machine M run applications A and B efficiently?" is a more appropriate question than "Does machine M work well?"

Just as we talk about how a particular hardware architecture processes software applications, we can also talk about how a particular software application processes a set of high-level tasks (see Figure 2-1). That is, we can think of a software application as also having an architecture. If we want to create documents or organize spreadsheets (two common sets of high-level tasks), we can choose among word-processing or spreadsheet applications. As in the hardware example, there are many possible application programs, with different programs designed to be optimal under different assumptions about the tasks. Which particular application architecture we choose dictates which subtasks will be easy and efficient. If, for example, our set of high-level tasks is writing the chapters in a calculus book, we are likely to choose a word-processing program that has functions and commands for formatting mathematical equations. If we are simply interested in writing letters, those functions are not important.

There are two common threads that underlie these examples of architecture. The first is the idea that for any level of a complex system, we can make a distinction between the fixed set of mechanisms and structures of the architecture itself, and the content those architectural mechanisms and structures process. So, it is fixed hardware that processes software content at

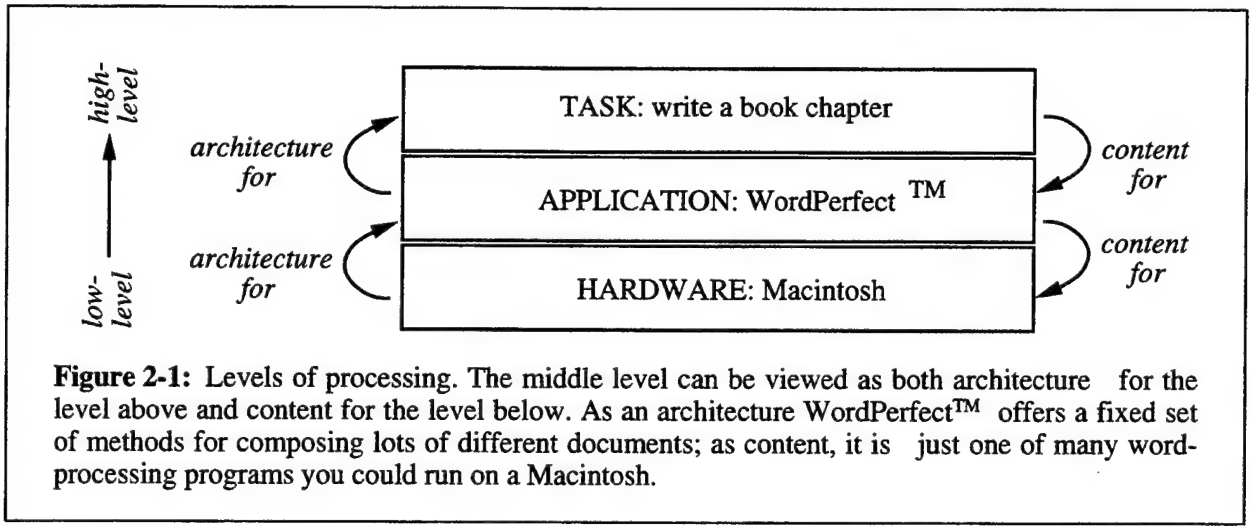


Figure 2-1: Levels of processing. The middle level can be viewed as both architecture for the level above and content for the level below. As an architecture WordPerfect™ offers a fixed set of methods for composing lots of different documents; as content, it is just one of many word-processing programs you could run on a Macintosh.

one level, and fixed application mechanisms that process high-level task content at the next. Another way of viewing this relationship is to note that an architecture by itself does nothing; it requires content to produce behavior. Because we'll have occasion to revisit this point periodically, it's worth being very clear:

BEHAVIOR = ARCHITECTURE x CONTENT

The second common thread running through the examples is that any particular architecture reflects assumptions on the part of the designer about characteristics of the content the architecture will process. There are, for example, many possible hardware architectures that process the same software, but machines with parallel processors will execute some software much more quickly than serial machines. Similarly, many applications will execute the same high-level task, but some provide single commands that achieve whole subtasks, while others require the user to construct sequences of commands to gain the same effect.

In general, then, the idea of architecture is useful because it allows us to factor out some common aspects of the wide array of behaviors that characterize the content. A particular architecture, that is, a particular fixed set of mechanisms and structures, stands as a theory of what is common among much of the behavior at the level above it.[1] Using this idea, we can

[1]The view of architecture as a theory of what is common across cognitive behaviors can be interpreted narrowly or broadly. In a modular architecture, for example, what is common to language might not be what is common to problem solving; each module would have its own architecture, its own factoring of what is common to the behaviors of that module. Our own approach, however, is to look for a minimal set of architectural components that is, as much as possible, common to all cognitive behavior. Thus, we view commonality broadly, both in our research and in this chapter.

define a *cognitive architecture* as a theory of the fixed mechanisms and structures that underlie human cognition. Factoring out what is common across cognitive behaviors, across the phenomena explained by microtheories, seems to us to be a significant step toward producing a unified theory of cognition. Thus, for most of the remainder of this chapter we will concentrate on this aspect of Soar, i.e. Soar as a cognitive architecture. As an example of a cognitive architecture, Soar is one theory of what is common to the wide array of behaviors we think of as intelligent. It is not the only such theory (see, e.g., (Anderson, 1993; Kieras & Meyer, 1994), but it is the one we will explore in detail.

In the sections that follow, we motivate the different structures and mechanisms of the Soar architecture by exploring its role in one concrete example of intelligent behavior. Because we are interested in a computationally-realizable theory, our exploration takes the form of constructing a computational model. To construct a model of behavior in Soar we must first understand what aspects of the behavior the architecture will support directly (Section 3), and lay the groundwork for tying the architecture to our sample content (Section 4). Then, piece by piece, we construct our model, introducing each architectural structure and mechanism in response to questions that arise from the behavior we are trying to capture (Sections 5 through 9). In Section 10 we step back and reflect on the architecture as a whole. Although we examine only one architecture and only one model based on that architecture, our broader goal is to provide both an appreciation for the power of the idea of cognitive architecture, and the tools to understand other architectures. Finally, in Section 11, we reflect briefly on where Soar stands as a candidate UTC.

3. What cognitive behaviors have in common

To understand how any computational architecture works, we need to use it to model some behavior (remember, the architecture alone doesn't do anything). What sorts of behavior should we model with Soar? A cognitive architecture must help produce cognitive behavior. Reading certainly requires cognitive ability. So does solving equations, cooking dinner, driving a car, telling a joke, or playing baseball. In fact, most of our everyday behavior seems to require some degree of thinking to mediate our perceptions and actions. Because every architecture is a theory about what is common to the content it processes, Soar is a theory of what cognitive behaviors have in common. In particular, the Soar theory posits that cognitive behavior has at least the following characteristics (Newell, 1990):

1. **It is goal-oriented.** Despite how it sometimes feels, we don't stumble through life, acting

in ways that are unrelated to our desires and intentions. If we want to cook dinner, we go to an appropriate location, gather ingredients and implements, then chop, stir and season until we've produced the desired result. We may have to learn new actions (braising rather than frying) or the correct order for our actions (add liquids to solids, not the other way around), but we do learn rather than simply act randomly.

2. **It reflects a rich, complex, detailed environment.** Although the ways in which we perceive and act on the world are limited, the world we perceive and act on is not a simple one. There are a huge number of objects, qualities of objects, actions, and so on, any of which may be key to understanding how to achieve our goals. Think about what features of the environment you respond to when driving some place new, following directions you've been given. Somehow you recognize the real places in all their detail from the simple descriptions you were given, and respond with gross and fine motor actions that take you to just the right spot, although you have never been there before.

3. **It requires a large amount of knowledge.** Try to describe all the things you know about how to solve equations. Some of them are obvious: get the variable on one side of the equal sign, move constant terms by addition or subtraction and coefficients by multiplication or division. But you also need to know how to do the multiplication and addition, basic number facts, how to read and write numbers and letters, how to hold a pencil and use an eraser, what to do if your pencil breaks or the room gets dark, etc.

4. **It requires the use of symbols and abstractions.** Let's go back to cooking dinner. In front of you sits a ten-pound turkey, something you have eaten but never cooked before. How do you know it's a turkey? You have seen a turkey before but never *this* turkey and perhaps not even an uncooked one. Somehow some of the knowledge you have can be elicited by something other than your perceptions in all their detail. We'll call that thing a symbol (or set of symbols). Because we represent the world internally using symbols, we can create abstractions. You can't stop *seeing* this turkey, but you can *think* about it as just *a* turkey. You can even continue to think about it if you decide to leave it in the kitchen and go out for dinner.

5. **It is flexible, and a function of the environment.** Driving to school along your usual route, you see a traffic jam ahead, so you turn the corner in order to go around it. Driving down a quiet street, a ball bounces in front of the car. While stepping on the brakes, you glance quickly to the sidewalk in the direction the ball came from, looking for a child who might run after the ball. As these examples show, human cognition isn't just a matter of thinking ahead, it's also a matter of thinking in step with the world.

6. **It requires learning from the environment and experience.** We're not born knowing how to tell a joke, solve equations, play baseball, or cook dinner. Yet, most of us become proficient (and some of us expert) at one or more of these activities and thousands of others. Indeed, perhaps the most remarkable thing about people is how many things they learn to do given how little they seem to be born knowing how to do.

There are many other properties that underlie our cognitive capabilities (for example, the quality of self-awareness), and there are other ways to interpret the same behaviors we have assigned to the categories above. What does it mean for Soar as an architecture to reflect this particular view of what is common to cognition? It means that the mechanisms and structures we put into Soar will make this view easy to implement, and other views more difficult. After we

have constructed our model within this architecture, it will be easy to describe the model's behavior as goal-oriented because the architecture supports that view directly. Any theory establishes a way of looking at a problem. If the theory is useful, it allows us to model the behaviors we want to model in a way that seems easy and natural.

Having identified some common properties of cognitive behavior that Soar must support, we should be able to motivate the specific structures and mechanisms of the architecture by tying them back to these properties. Keep in mind our equation:

BEHAVIOR = ARCHITECTURE x CONTENT

It's clear that if we want to see how the architecture contributes to behavior then we need to explore the architecture in terms of some particular content. For example, describing how Soar supports the underlying characteristic of being goal-oriented won't by itself produce behavior; we have to be goal-oriented *about* something. Let's consider a simple scenario from baseball[2]:

> Joe Rookie is a hypothetical pitcher with the Pittsburgh Pirates, about to throw the first pitch of his major league career. He chooses to throw a curve ball. The batter, Sam Pro, hits the ball, but Joe is able to catch it after it bounces once between home plate and the pitching mound. Then, he quickly turns and throws the batter out at first base.

Figure 3-1 shows the participants in our scenario in their usual locations and roles, and the path of the ball from Joe's pitch to the out at first. Typical rhetoric about baseball players aside, it is clear that Joe displays many of the characteristics of intelligent behavior in our simple scenario. In particular, he:

1. **Behaves in a goal-oriented manner.** Joe's overriding goal is to win the game. In service of that goal, Joe adopts a number of *subgoals* — for example, getting the batter out, striking the batter out with a curve ball, and when that fails, throwing the batter out at first.

2. **Operates in a rich, complex, detailed environment.** As Figure 3-1 shows, there are many relevant aspects of Joe's environment he must remain aware of throughout the scenario: the positions and movement of the batter and the other members of his team, the number of balls and strikes, the sound of the bat striking the ball, the angle his body makes with the first baseman as he turns to throw, etc.

---

[2]This chapter is based on the introductory lecture of a tutorial on Soar given to incoming graduate students at Carnegie Mellon University in the fall of 1993. The playoffs and approaching World Series were often in the news, creating a timely backdrop for the example. For those unfamiliar with baseball, it is a contest between opposing sides who try to score the most runs within the fixed time frame of nine innings. To score a run a batter stands at home plate and tries to hit a ball thrown by the opposing team's pitcher in such a way that he and his teammates will advance sequentially around the three bases and back to home plate (see Figure 3-1). There are a number of ways the team in the field may halt the batter's progress, resulting in an out. An inning is over when each side has been both at bat and in the field for the duration of three outs. For a complete description of the game, see (The Commissioner of Baseball, 1994).

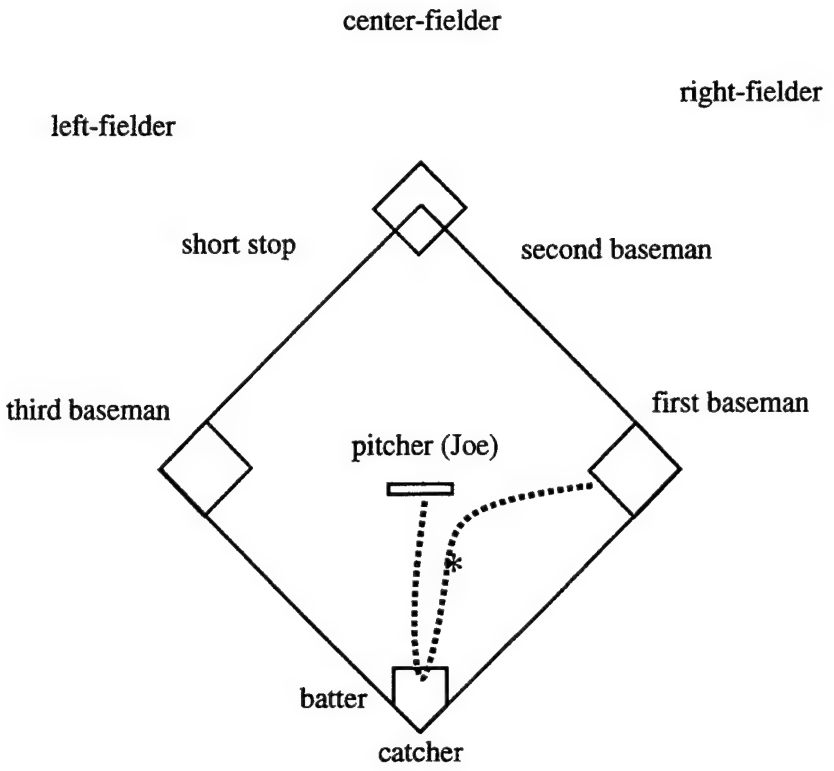


Figure 3-1: Joe's curve ball is hit by the batter, caught by Joe on a single hop and thrown to first base.

3. **Uses a large amount of knowledge.** In deciding on his pitch, Joe probably draws on a wealth of statistics about his own team, his own pitching record, and Sam Pro's batting record. We discuss Joe's knowledge in more detail below.
4. **Behaves flexibly as a function of the environment.** In choosing his pitch, Joe responds to his own perceptions of the environment: Is it windy? Is the batter left- or right-handed? etc. Although not included in our scenario, he may also have to consider the catcher's suggestions. When the ball is hit, Joe must show flexibility again, changing his subgoal to respond to the new situation.
5. **Uses symbols and abstractions.** Since Joe has never played this particular game (or even in this league) before, he can draw on his previous experience only by abstracting away from this day and place.
6. **Learns from the environment and experience.** Learning is the acquisition of knowledge that can change your future behavior. If he's going to stay in the major leagues, Joe had better learn from this experience, and next time throw Sam a fast ball.

Just as the architecture is a theory about what is common to cognition, the content in any particular model is a theory about the knowledge the agent has that contributes to the behavior. For our model of Joe to act like a rookie pitcher, we will have to give it many different kinds of

knowledge, some concrete examples of which are shown in Table 3-1.

K1: Knowledge of the objects in the game e.g. baseball, infield, base line, inning, out, ball/strike count K2: Knowledge of abstract events and particular episodes e.g. how batters hit, how this guy batted last time he was up K3: Knowledge of the rules of the game e.g. number of outs, balk, infield fly K4: Knowledge of objectives e.g. get the batter out, throw strikes K5: Knowledge of actions or methods for attaining objectives e.g. use a curve ball, throw to first, walk batter K6: Knowledge of when to choose actions or methods e.g. if behind in the count, throw a fast ball K7: Knowledge of the component physical actions e.g. how to throw a curve ball, catch, run

Table 3-1: A small portion of the knowledge needed to model Joe Rookie.

Neither the architecture nor the content alone produce behavior. Before our model can throw its first pitch we must find some way to process Joe's knowledge in Soar. How can we express the knowledge so that our model of Joe acts in a goal-oriented way? To answer this question we introduce the idea of behavior as movement through a *problem space* in the next section.

4. Behavior as movement through problem spaces

Standing on the mound in the hot sun, Joe has a difficult goal to achieve and many ways to achieve it. To start, he must throw one of the many pitches in his repertoire. Once thrown, any of these pitches might result in a strike, a ball, a single, double, triple, or a homerun, and so on. Under some of these conditions Joe will have to pitch again to Sam Pro, under others he will face a different batter. If he faces a different batter, Sam may be on base or not; each of the variations in outcome is relevant to which pitch Joe chooses next. We can graphically represent the space of possible actions for Joe, as he pursues his goal of getting Sam Pro out, as in Figure 4-1. The text in the picture describes the situations that Joe might be in at different moments in time. The arrows represent both mental and physical actions Joe might take to change the situation, like deciding on a pitch, throwing a pitch, and perceiving the result.

If we try to imagine (and draw) *all* the choices Joe might have to make during a game, given *all* the circumstances under which they might arise, we are quickly overwhelmed. Of course, even though Joe is able to make a choice in any of the circumstances described, he makes only

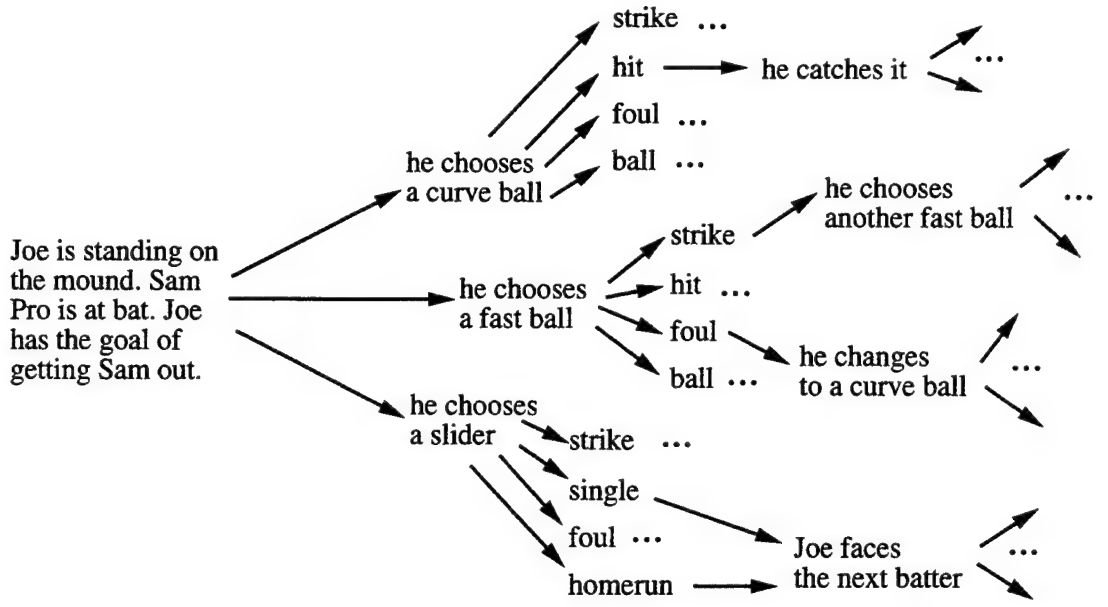


Figure 4-1: Looking at Joe's behavior as a space of decisions made over time. From the initial situation of holding the ball, Joe must decide what pitch to throw. On the basis of the outcome, he will make his next choice, and so on.

one choice at a time while playing the actual game. In terms of Figure 4-1, Joe will actually decide on and throw only one first pitch of the game, it will either be hit or not, and so on. Joe must make his decisions with respect to the situation at the moment — based on that situation, what is recalled about the past, and what can be anticipated about the future — but does so over all the moments of his life. What this means for our model of Joe is that it, too, must be able to act based on the choices that make sense at the moment, but it must also be able to act under all the moments that may arise.

Figure 4-1 is an example of viewing behavior as movement through a *problem space* (Nilsson, 1971; Newell & Simon, 1972). The idea of problem spaces dates back to the earliest days of artificial intelligence research and cognitive modeling using computers, and is one of the central computational constructs in the Soar theory of cognition. It may be hard to directly see problem spaces in the brain when focusing on it as a mass of neurons, just as it may be hard to see this book when using a microscope to examine the atoms out of which it is composed; however, the theory states that problem spaces are structures at some higher level of organization of the brain's neurons, just as this book is a structure at some higher level of organization of a set of atoms. For the purpose of building a model, the representation in Figure 4-1 is particularly

useful because it supports two points of view: a static view of Joe's life, which we can use to talk about all the possible actions he might take in a particular situation, and a dynamic view of Joe's life, which we can use to talk about the actual path his behavior moves him along.

The abstract form of a problem space is depicted graphically in Figure 4-2. The problem space itself is represented by a triangle to symbolize the ever-expanding set of possibilities that could unfold over time. Some of these possibilities will be represented explicitly, as explained below, while others are represented only implicitly, by ellipsis (...). The goal is represented by a circle at the apex of the triangle. Notice how everything in the problem space in Figure 4-1 was pertinent to Joe's goal of getting Sam out. One of the fundamental ideas behind problem spaces is that they partition knowledge in goal-relevant ways.

The situations described in text in Figure 4-1 are represented in the general case by *states* (squares in Figure 4-2). A state is described in terms of a vocabulary of features (in bold face) and their possible values (in italics). (The value of a feature may itself be a set of features and values, making possible the representation of a situation as richly interconnected sets of objects, as we shall see below.) So we might represent the part of the situation in Figure 4-1 described by the text "Sam Pro is at bat," by a pair of features and values (e.g. **batter name** *Sam Pro* and **batter status** *not out*), and these features and values would be part of any state intended to represent that part of the situation. In our examples so far we have been using features, values, and states to represent things in the observable situation. But there are always internal, non-observable aspects of any situation as well (e.g. Joe's perceptions, or things Joe may need to reason about that have no physical correlate, like batting average). The state is a representation of *all* the aspects of the situation — internal and external — that the model may need to choose its next action.

To model a particular behavior, there must be an initial description of the situation, or *initial state* (S_0). There must also be a description that corresponds to the desired *goal state* or states (shaded squares), i.e. those features and values that indicate that the goal has been achieved. In Figure 4-2, the goal states are exactly those in which feature **f2** has value *v6*, regardless of the other features and their values. If **f2** is **batter status** and *v6* is *out*, then the goal states in Figure 4-2 would correspond to our informal notion of Joe having achieved his goal of getting Sam Pro out.

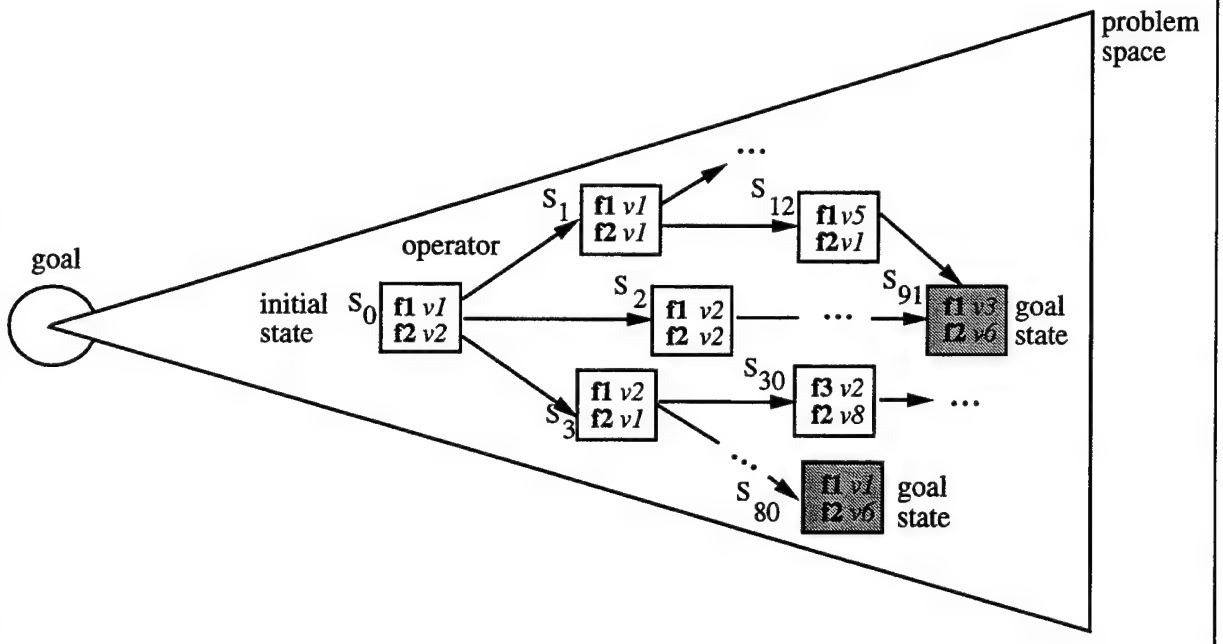


Figure 4-2: Behavior through time represented as movement through a problem space. The problem space is represented by a triangle to symbolize the ever-expanding set of possibilities that could unfold over time. The goal is represented by a circle at the apex of the triangle. Squares represent states, the features and values that reflect the internal and external situation. Goal states, states in which features have values that indicate the goal has been achieved, are shaded. Arrows represent operators that change or transform states. A state transformation may correspond to internal behavior (conscious or unconscious) or external behavior (actions observable in the world), or both. The states, features (bold face), and values (italics) are identified with arbitrary symbols, e.g. S1, **f1** *v2*, for convenience.

Both internal behavior (conscious or unconscious mental actions) and external behavior (actions observable in the world) correspond to moving along a path that leads from the initial state to a goal state via *operators* (arrows). At every point in time, there is a single state designated as the *current state* because it represents the current situation. Movement from the current state to a new state occurs through the application of an operator to the current state; an operator transforms the current state by changing some of its features and values. The application of an operator may cause only one feature value to change (e.g. S_0 to S_1) or may change multiple features and values (e.g. S_3 to S_{30}).

In essence, a problem space is defined by a particular set of states and operators, relative to a goal. Of course, movement through a problem space could be entirely random. To keep behavior goal-directed, the succession of operators that are applied to the state and the resulting state transformations must be guided by *the principle of rationality*: if an agent has knowledge

that an operator application will lead to one of its goals then the agent will select that operator. This may seem like a simple idea, but we will see that it pervades every aspect of the Soar architecture.

We are now in a position to answer the question posed at the end of the previous section. How can we express the different kinds of knowledge the model must have so that it acts in a goal-oriented way? The answer is: by representing the knowledge in terms of states and operators and guiding the choice of which operator to apply by the principle of rationality. Rewriting our scenario in these terms, we say that Joe has the goal of getting the batter out in the state defined by the start of the game. Of all the operators available for each state in the space, he should choose the one that he thinks will transform the current state to a new state closer to the desired goal state.

Describing the underlying idea of problem spaces puts us on the path to our own goal of modeling Joe Rookie in Soar. Mapping knowledge into states and operators is the first step toward tying the content of Joe's world to the Soar architecture. Not surprisingly, this first step raises a new set of questions that we, as model builders, must answer: What knowledge becomes part of the state and what knowledge becomes part of the operators? How do we avoid having every operator available in every state? How do we know what an operator application will do? How do we know when the goal has been achieved? To answer these questions, the next section introduces the first of our architectural structures, the goal context.

5. Tying the content to the architecture

Consider Table 3-1 again. The list incorporates many different kinds of knowledge that our model must include: knowledge about things in the world (K1 and K2) and knowledge about abstract ideas (K3 and K4), knowledge about physical actions (K7) and knowledge about mental actions (K5), even knowledge about how to use the other kinds of knowledge (K6). This content is expressed in the familiar, everyday terms of the *domain* of baseball. But the architecture of cognition cannot be designed for baseball alone. Because the architecture must support what is common across many domains, its mechanisms must process a domain-independent level of description. In particular, Soar's architectural mechanisms process the four kinds of objects introduced in the previous section: goals, problem spaces, states, and operators. A particular instance of these four kinds of objects is organized into a structure called a *goal context*, which

contains four ''slots,'' one for each of the objects.

Figure 5-1 presents a goal context for Joe Rookie graphically. It shows that Joe has the goal (circle) of getting the batter out. He is trying to achieve that goal using the Pitch problem space (triangle). At the moment, he is in a state (square) in which the batter, Sam Pro, is not out, and he has selected the throw-curve operator (arrow) to apply to this state to produce a new state.

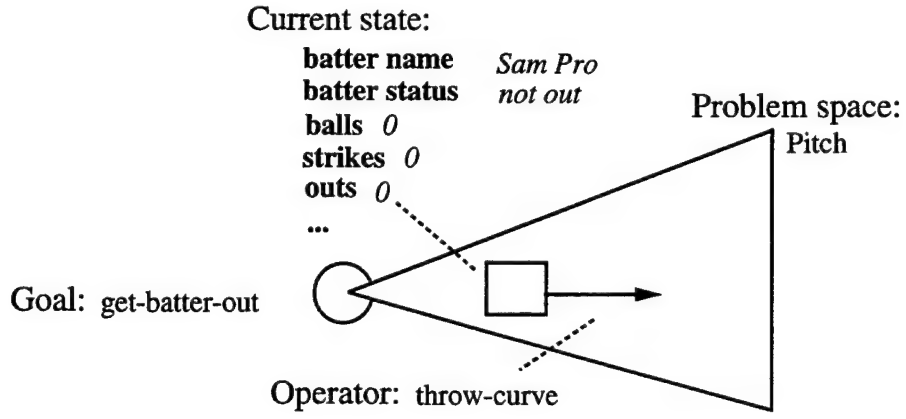


Figure 5-1: A goal context for Joe.

If we look at Table 3-1 and Figure 5-1 together, some simple guidelines for tying domain content to the goal context emerge. Roughly speaking, we use knowledge of objectives (K4) for our goals and knowledge about actions (K5 and K7) to define our operators. The domain knowledge of the objects and people in the game (K1) is represented by the features and values in the state.

Remember that a problem space is simply a set of possible states and operators that may occur in trying to reach a goal. By organizing the domain knowledge in a Soar model into multiple problem spaces, we limit the set of operators to be considered in searching for a goal's desired state. In other words, the knowledge of objectives also helps to answer the question, ''How do we avoid having every operator available in every state?'' Because the model is always in a state within a problem space, it will be limited to considering the operators available in that problem space at that state. In the Pitch problem space the operators will be the various kinds of pitches Joe can throw. So in the context of trying to get the batter out (the goal) by pitching (the problem space), the model considers only pitch operators. It does not consider, for example, operators for

fielding or batting or doing laundry; it behaves in a goal-oriented way. Further limits to which operators to consider come from content knowledge of when to choose actions or methods. In Table 3-1, this sort of knowledge (K6) enables the model to choose a curve ball based on particular attributes of the state, such as whether Sam Pro is a left- or right-handed batter. (In Section 7, we will consider what happens when the knowledge available to the model in a problem space is inadequate to choose a single operator.)

Assume that the model of Joe has chosen to throw a curve ball; how do we know what that (or any other) operator will do? There are really two answers to this question, each suggested by a different source of knowledge. On the one hand, what the operator will do may be defined completely by the execution of the operator in the external world. In our case, imagine our model of Joe as a baseball-playing robot. If knowledge of physical actions is used (K7), the application of the throw-curve operator can result in appropriate motor actions by the robot. Once the motor actions have been performed (and the results perceived), both the world and the model will be in new states. On the other hand, what the operator will do may be defined by knowledge of abstract events or particular episodes (K2). Imagine that Figure 5-1 is a snapshot of the model's processing while it is evaluating what to do next. In this case, the application of the throw-curve operator should not result in an action in the world. Instead, the next state should depend on the model's knowledge about curve balls in general, or about what happens when Sam Pro is thrown curve balls. Clearly, the outcome of applying the operator as a mental exercise is a state that may or may not resemble the state that results from throwing the curve ball in the world. What is important is not whether the operator application preserves some kind of truth relation between the external world and its internal simulation. What is important is that a single structure, the goal context, allows us to model either behavior — acting or thinking about acting — as a function of what kind of knowledge is brought to bear.

Imagine now that our model of Joe has thrown the ball, either in the world or in its head. How do we know if the resulting state has achieved the goal of getting the batter out? In the domain of baseball, determining that the current state is a desired state relies on knowledge of the rules of the game (K3).

In this section we have seen how domain content can be tied to an architectural construct, the goal context. The content-independent specification of goals, problem spaces, states and operators is key to their usefulness. If the architecture is going to support what is common

across domains, its behavior can't be limited by the structure or content of any one domain. Although we have demonstrated how different kinds of knowledge map naturally to different aspects of the goal context, we have left quite a few important details unspecified. In particular: How should the knowledge in its general form be represented? How should it be represented within a goal context? What are the mechanisms for perceiving and acting on the external world? And most important: What are the architectural processes for using knowledge to create and change the goal context?

6. Memory, perception, action, and cognition

To make sense of our everyday experience and function in the world, we need to use our knowledge about objects and actions in pursuit of our current goals. You know many things about a cup but you use only some of that knowledge when you want a cup of coffee, and some other part of it when you want a place to put pens and pencils. Moreover, once you have applied some of that knowledge, a particular cup will be full of coffee or full of pens. Your ability to reason about other cups as pencil holders or coffee mugs will not have changed but there will be something in your current situation that will have taken on a single role — your general knowledge will have been tied to the particular instance of the cup in front of you. This dichotomy between general knowledge and specific applications of that knowledge is captured in Soar by the existence of two different memory structures. Knowledge that exists independent of the current goal context is held in the architecture's *long-term memory* (LTM). The situation described by the particular current occurrence of some portion of that knowledge is held in Soar's *working memory* (WM). To say this a little more intuitively, it is useful to think about LTM as containing what can be true in general (e.g., cups can hold coffee or pencils), and WM containing what the model thinks is true in a particular situation (e.g., the cup in front of me contains coffee).[3]

As we know from the previous section, behavior can occur only after we tie the content of the domain to a goal context. We certainly want to put the content of the domain in long-term memory, where it can be expressed in general terms, independent of any particular situation. The

---

[3]Strictly speaking, LTM can also contain particular situations, i.e. episodic memories like "the pencils in my coffee cup yesterday." We discuss this type of association in Section 8, although, for the moment, the simpler characterization given above will suffice. In addition, WM can also contain features and values that define general rules (e.g. **hint-for-tic-tac-toe** *take the center square first*), though these must be interpreted by associations in LTM to result in behavior.

goal context, then, resides in working memory. In fact, the structure of working memory is exactly the four slots of the goal context that hold the values for the current goal, problem space, state, and operator, as well as the features and values that make up the content of the state (see Figure 5-1). Knowledge moves from its general form in LTM to its specific form in WM by a process called *the decision cycle*. Let's consider each memory structure and process in turn.

We begin with long-term memory where knowledge is represented as associations. Table 6-1 shows some examples of associations needed for a model of Joe Rookie. The exact format of the knowledge as it is given to Soar is unimportant; for expository purposes, we write the knowledge as English-like if-then rules. Such a rule represents an association between a set of conditions, expressed in terms of features and values and specified by the "if" part of the rule, and a set of actions, also described by features and values, in the "then" part. As in Figure 5-1, when describing information that is part of the state, we use boldface for features and italics for values.

Since working memory is just the goal context, Figure 5-1 serves as an example. We can use this figure and Table 6-1 together to make some preliminary observations about the relationship between LTM and WM. First, note that the "if" portion of each LTM association tests either a perception or elements of the goal context in WM: the goal, problem space, state (and its substructure of features and values), or operator. If there is a match between the "if" part of an association and elements in WM, we say that the association has been *triggered.* This, in turn, causes the "then" portion to *fire* by either sending a message to the motor system (a7) or suggesting changes to the goal context. Thus, each matching association maps from current goal context elements to new goal context elements.[4] Second, note that there can be dependencies between the associations. For example, a2 will not match the current context until a1 has matched and fired, and a4 will not match until a1 through a3 have matched and fired. These dependencies are part of the semantics of the domain of baseball — you don't choose a pitch until you've decided to pitch to the batter and you don't pitch to a batter if you aren't the pitcher on the mound. Soar doesn't recognize the existence of these dependencies: it simply matches the "if" portions and performs the "then" portions. Because all the elements of the associations are expressed in terms of perceptions, actions, and goal context elements, they are processed by

[4]We are using the word *map* in its mathematical sense, where it refers to a type of function. A simple function, such as Y = f(X), specifies a single value of Y for each value of X. In contrast, a mapping function can return many values. In this sense, the LTM associations in Soar are mapping functions because, given a particular set of WM features and values, a matching LTM association can return many new features and values to WM.

(a1) If I perceive I am at the mound
then suggest a goal to get the batter out
(a2) If there is a goal in WM to get the batter out
then suggest achieving it using the Pitch problem space with
an initial state having **balls** *0* and **strikes** *0*
(a3) If using the Pitch problem space
and I perceive a new batter who is left/right handed
then add **batter** *not out* and **batter** *left/right-handed* to the state
(a4) If using the Pitch problem space and the **batter** is *not out*
then suggest the throw-curve operator
(a5) If using the Pitch problem space and the **batter** is *not out*
and the **batter** is *left-handed*
then suggest the throw-fast-ball operator
(a6) If both throw-fast-ball and throw-curve are suggested
then consider throw-curve to be better than throw-fast-ball
(a7) If the throw-curve operator has been selected in the Pitch problem space
then send throw-curve to the motor system and add **pitch** *thrown* to the state
(a8) If using the Pitch problem space and the **pitch** was *thrown* and I perceive a hit
then add **pitch** *hit* to the state

Table 6-1: Some of the associations in the model's long-term memory. In the decision cycle, patterns of percepts and state features and values in WM are matched to the "if" portion of an association in LTM, and changes to the WM state and motor actions are produced by the associated "then" portion.

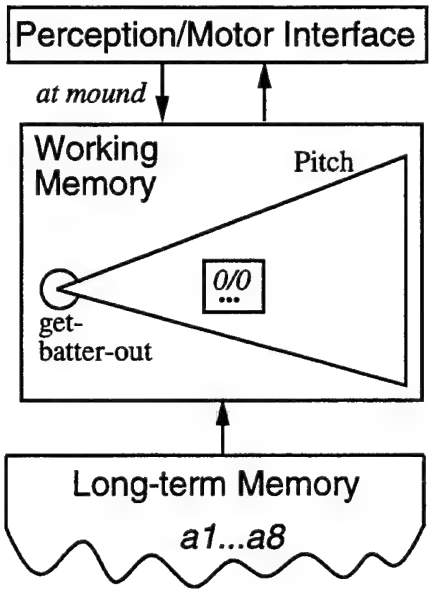
the architecture in a completely general, domain-independent way. Put slightly differently: if you can express a domain's content in these terms, the architecture can use that content to produce behavior.

Unlike the cognitive processes it represents, a model begins and ends its processing at arbitrary points in time. A human Joe Rookie does not suddenly find himself on the pitching mound with no context to explain how he got there, but our model of Joe begins with no elements in working memory. Working memory elements in Soar arise in one of two ways: through simulated

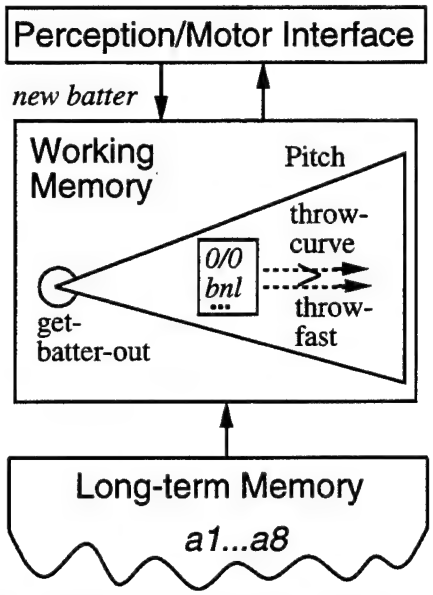
perception or through associations that fire during the decision cycle. The goal of getting the batter out, for example, is established in working memory when association a1 fires because the Soar model of Joe perceives that it is on the pitching mound. Modeling perception is a complex topic which we will, for the most part, not address. What is important for our purposes here is that the architecture provides a straightforward facility (the Perception/Motor Interface) for introducing elements into working memory from the external environment via perception, whether that perceptual input comes from a simple simulation program, the output from a video camera, or feedback from a robot arm. Perception in Soar is an asynchronous process with respect to cognition. In other words, perceptual input enters working memory (as additional feature-value information added to the state) irrespective of the decision cycle, which we will describe next.

The decision cycle is the processing component that generates behavior out of the content that resides in the long-term and working memories. The purpose of the decision cycle is to change the value in one of the four slots of the goal context. Remember the problem space point of view discussed in Section 4: goal-directed behavior corresponds to movement in a problem space from the current state to a new state through the application of an operator to the current state. Concretely, then, the purpose of the decision cycle is most often to change the value in the operator slot of the goal context. To understand how the decision cycle works, let's focus on how the model chooses and throws the curve ball. Figure 6-1(a) shows the state of working memory after the decision cycles in which associations a1 and a2 fired, establishing the get-batter-out goal, the Pitch problem space, and the initial state which contains the substructure for the count (abbreviated as 0/0 for **balls** *0* and **strikes** *0*). The ellipsis in the state represents other working memory elements resulting from perception.

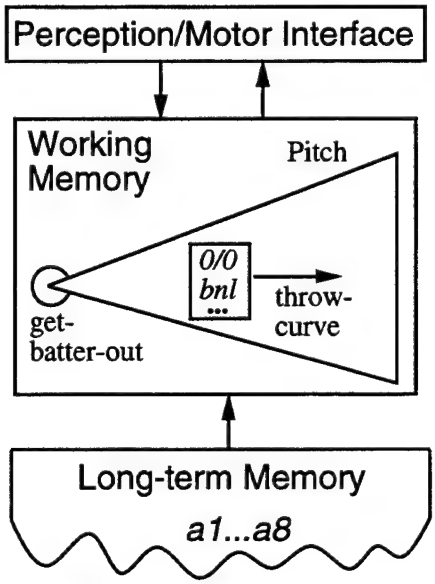
A decision cycle is a fixed processing mechanism in the Soar architecture that does its work in two phases: *elaboration* and *decision*. During elaboration, the contents of working memory are matched against the "if" parts of the associations in long-term memory. All associations that can fire do fire, in parallel, resulting in changes to the features and values of the state in addition to suggestions, or *preferences*, for changes in the context slots. As a result of the working memory changes, new associations may fire. Elaboration continues in parallel waves of association firings until no more associations fire. Figure 6-1(b) shows the results of the elaboration phase for our example. First, the appearance of a new batter in a particular



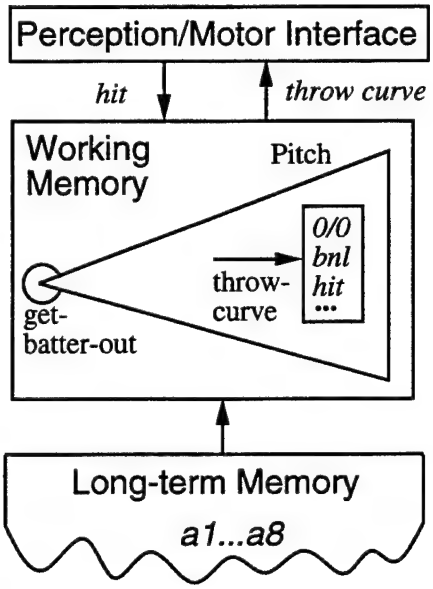
(a) Decision cycles 1& 2: creation of goal, problem space and initial state via a1&a2.



(b) The elaboration phase of cycle 3 results in state changes via a3, then operator proposals via a4-a6.



(c) The decision phase of cycle 3 resolves the preferences for operator proposals and selects the throw-curve as the new value for the operator context slot.



(d) The elaboration phase of decision cycle 4 results in the application of the throw-curve operator via a7 and subsequent state changes via a8.

Figure 6-1: Selecting and throwing the curve ball. In the first wave of the elaboration phase, the context in (a) triggers a3, adding information about the batter's handedness to the state. This change then triggers associations a4 and a5 in the second wave, producing proposals for throw-curve and throw-fast-ball. The operator proposals trigger a6 in the third wave, producing a preference for throwing the curve ball. No more associations fire so Soar enters the decision phase (c). The result of the decision procedure is to select the throw-curve operator for the operator slot of the goal context. The changes to the state that result from applying this operator actually occur during the elaboration phase of the next decision cycle (d).

orientation to home plate causes association a3 to fire, augmenting the state with information about the batter's presence and handedness (*bnl* in the figure, which is shorthand for **batter** *not out* and **batter** *left-handed*). If we assume Sam Pro is left-handed, then in the next wave of elaborations, associations a4 and a5 fire in parallel, suggesting both a curve ball and a fast ball as pitches (dotted arrows). These suggestions cause a6 to fire in the next elaboration, augmenting working memory with a preference for the curve ball (represented by the greater than sign over the dotted arrows). Because no more of the associations in LTM fire, the elaboration phase is complete.

At the end of the elaboration phase, two operators, throw-curve and throw-fast-ball have been suggested as candidates for the operator slot in the context. Why are changes to the context slots phrased as suggestions or preferences? To make sure all the knowledge available in the current context is brought to bear before a decision is made. That is, a decision to change a context slot should not be made until all the associations in LTM that match the current context have been retrieved. Remember the principle of rationality: the model should use its knowledge to achieve its goals. The introduction of preferences allows the architecture to collect all the available evidence for potential changes to the context before actually producing any change. Thus, quiescence in elaboration, i.e., the absence of any further firings of LTM associations, signals the start of the decision phase. This is when the preferences added to working memory are evaluated by a fixed architectural decision procedure. The decision procedure is applied to the vocabulary of preferences and context slots, independent of the semantics of the domain. In other words, the decision procedure isn't written in terms of throw-curve and throw-fast-ball; it's written in terms of operators and preferences among operators. In our example, there are suggestions for two operators and a "better" preference, suggested by association a6, that orders their desirability. So, as shown in Figure 6-1(c) the outcome of the decision procedure is to choose the suggested operator that is better: throw-curve. Remember that the result of the decision cycle is the change of a single context slot — either a new goal, a new problem space, a new state or a new operator is selected. So, the outcome of this decision cycle is the addition of the selected operator to the context.

Of course, selection of the operator isn't enough. The operator must be applied to produce the state transition that completes a move in the problem space. The application of the operator occurs during the elaboration phase of the next decision cycle, as shown in Figure 6-1(d). The

throw-curve operator in working memory causes association a7 to fire, producing a motor command to be sent by the Perception/Motor Interface to the external environment. In turn, Sam hits the curve ball, an event that is perceived by the model and encoded as an augmentation to the state via association a8.

Given the limited knowledge we have put into long-term memory, Figure 6-1(d) represents the end of the behavior we can elicit from the model. Yet the answers to the questions we asked at the end of Section 5, and the basic outline of further processing, should be clear:

- How should general knowledge be represented? As associations that map one set of working memory elements into a new set of working memory elements. The working memory elements themselves are just features and values which, with the exception of the four special slots of the goal context, have no meaning to the architecture.
- How should knowledge be represented within a goal context? As the simple features and values that are the building blocks of the associations in long-term memory. The representations of knowledge in long-term memory and in the goal context are inextricably linked because the decision cycle is essentially a controlled process for matching elements in the context to the ''if'' patterns in LTM.
- What are the architectural processes for using knowledge in LTM to create and change the goal context? The decision cycle with its elaboration and decision phases. The two phases are necessary for the architecture to produce behavior in accordance with the principle of rationality. A result, however, is that the representation of knowledge in long-term memory must be cast in terms of preferences and suggestions. This must occur so that decisions are postponed until the moment when all knowledge has been elicited by the current context and can be integrated to produce a change to the context.
- What are the mechanisms for perceiving and acting on the external world? Perception and action go through an interface that runs asynchronously to cognition's decision cycle. Because percepts add new information to working memory, however, perception must ultimately produce working memory elements in the same representation as long-term memory, i.e. from the same vocabulary of features and values. Similarly, motor actions may be instigated at any point in the decision cycle, but the trigger for those actions will be working memory elements.

Considering our questions one at a time leads to answers that are somewhat circular because the basic structures and processes of the architecture must form a coherent whole. The decision cycle can only serve as an adequate process for ensuring goal-oriented behavior if long-term memory has a certain form. Working memory can only contribute to behavior that is flexible as a function of the environment if the decision cycle is essentially a domain-independent process of matching WM elements to the ''if'' parts of the LTM associations. Thus, each piece is motivated at least in part by the existence of the others. Together they serve as a simple, elegant theory of how the knowledge that Joe has can produce the behavior we see.

What we have shown in this section is how the model can be made to produce some of Joe's behavior by casting content in terms the architecture can manipulate. The knowledge that we described in everyday terms in Table 3-1 was redescribed as associations between contexts in Table 6-1. Once in this form, the main processing components of the architecture could be applied to the content, with the decision cycle mediating each move through the problem space. In the example, the knowledge in our domain content led directly to a single change in the goal context at the end of each decision cycle. But what happens if the decision cycle can't decide?

7. Detecting a lack of knowledge

Look again at Table 6-1, and consider what would happen if association a6 were not part of our domain content. At the end of the elaboration cycle in which a3, a4, and a5 fire, two operators would have been suggested with no preferences to decide between them. The architecture requires that a single operator be selected, but without a6, the knowledge that could be elicited from long-term memory by the current context would not be adequate to meet that constraint. In short, processing would reach an *impasse* because of the tie between the two proposed pitch operators. Indeed, in Soar, an impasse is an architectural structure that arises whenever the decision procedure cannot resolve the preferences in working memory to produce a single change in the context slots. An impasse is what happens when the decision cycle can't decide.

Now it might seem that if a6 should be part of our content but isn't, then its absence simply represents an inadequacy of the theory or poor design of the model. The impasse, however, actually has an important role to play in the architecture. Recall that Soar supports multiple problem spaces as a way of partitioning knowledge and limiting the operators to be considered in searching for a goal's desired state. So far, we've formulated our model with a single problem space, Pitch, but we have tacitly assumed that other problem spaces will be required to produce the full range of Joe's behavior. Clearly, if the current problem space is part of the context used to elicit knowledge from long-term memory, and there are multiple problem spaces, then a failure to elicit knowledge may mean only that the relevant knowledge is not associated with the current context. When an impasse arises, then, the architecture automatically establishes the goal of eliciting the knowledge needed to continue processing.

Figure 7-1(a) shows graphically the presence of the impasse (crescent) and the resulting goal

(resolve-tie) for our example. In this case, the impasse is an operator tie in the Pitch problem space, and the goal is simply to resolve the operator tie.

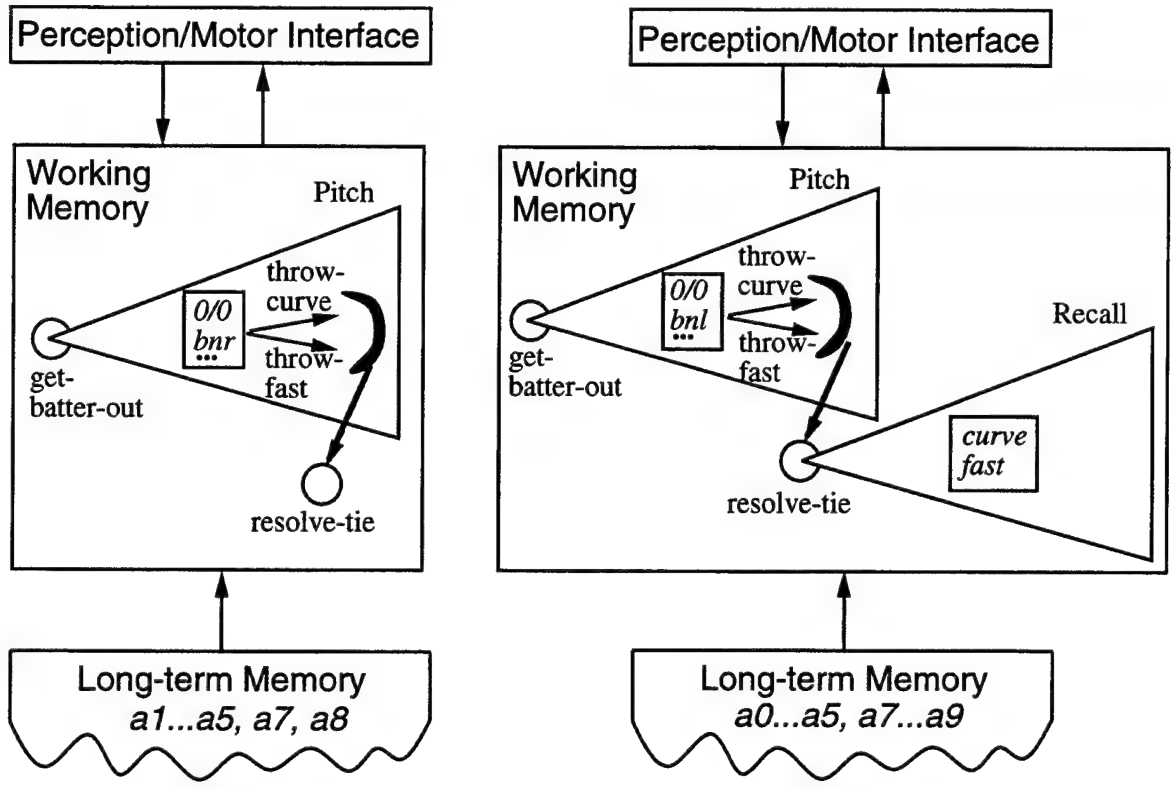


Figure 7-1: (a) An operator-tie impasse in Pitch causes a new context to arise. When the decision cycle can't decide between the two proposed operators, the architecture creates an impasse structure (crescent) which links the current goal (get-batter-out) to its subgoal (resolve-tie), producing a goal-subgoal hierarchy. (b) Knowledge in the domain content then determines the rest of the subgoal context, i.e. the problem space to work in to achieve the subgoal, the initial state and the operators that are applicable to it. The decision cycle may now result in a single change to either goal context in the hierarchy. In general, processing will continue in the lower problem space (Recall). If the decision cycle suggests a change to a slot in the higher goal context (Pitch), however, that change takes precedence and the lower context will disappear.

Since the architecture establishes only the subgoal when an impasse arises, our domain content must fill out the remainder of the new goal context using LTM associations that select a problem space, state, and operator. In other words, we must provide the model with some method for deciding between the pitches in order to resolve the operator-tie impasse. Let us assume that the way Joe would make this decision is by trying to remember how successful he's been with each of the pitches in the past. To link this bit of content to the architecture, we add the following

association to the model's LTM:

(a9) If there's a goal to resolve an operator tie in the Pitch problem space
then suggest achieving it using the Recall problem space with an initial
state containing the tied operators.

Figure 7-1(b) shows how the new goal context is filled out via association a9 in the decision cycle that follows the impasse. Once the new context has been established, all of the architectural mechanisms we have already discussed can be applied to it. So, this new context will trigger associations that result in operator proposals that, in turn, transform Recall's state, etc.

In general, then, working memory actually consists of a goal-subgoal hierarchy where each subgoal exists in order to resolve an impasse in the problem space above it.[5] The hierarchy grows as impasses arise in the normal course of processing, and shrinks as impasses are resolved by eliciting the knowledge required to make progress in the higher context. Typically, decision cycles correspond to movement through the problem space that is at the bottom of the hierarchy, but it is not necessary that this be true; the decision cycle still results in only a single change to one of the context slots but if multiple changes are suggested in different contexts, the change to the context that is highest in the hierarchy is the one that occurs. Of course, if a change occurs to a context high up in the hierarchy, then the knowledge needed to overcome the impasse in that context must have become available, so all the subcontexts in the hierarchy below the changed context, which were created because of the impasse, disappear.

An operator tie is just one of the ways in which the decision cycle can fail to decide. An operator no-change impasse, for example, occurs when knowledge about when to select an operator is available in one problem space, but knowledge about the implementation of the operator (how it changes the state) is not available in that space. For our purposes, what is important is not to understand the particulars of the different kinds of impasse, but to understand that the full set of impasses defined in Soar is fixed, and domain-independent. As with the vocabulary of preferences, the architecture defines a vocabulary of impasses so that it can process domain content without understanding the meaning of that content.

---

[5]In fact, every goal in Soar is established by the architecture, a fact we have glossed over in the preceding discussion for expository purposes. The highest level goal is created automatically; all subsequent goals result from impasses. Thus, the goal "get-batter-out" in Figures 5-1, 6-1, and 7-1 is really shorthand for the goal to resolve the impasse on the get-batter-out operator in a higher problem space that is not shown (but see Section 11).

Returning to our example, our model of Joe must still choose a pitch. Conceptually, the resolution of the operator-tie impasse can come about in two ways. If the catcher signals, for example, then information coming in through perception might change the state in Pitch and elicit preferences for one pitch or the other (or for some other behavior altogether). In this case, the get-batter-out context in the goal hierarchy would change in the next decision cycle, and the impasse would simply disappear. Alternatively, processing could continue in the lower context. Perhaps operators in Recall that elicit knowledge about how well Joe has done with each pitch in the past will resolve the impasse, or perhaps further subgoals will arise. In any of these cases, we must add more content to our model to have the model behave appropriately. By now it should be reasonably clear what form that knowledge should take and how to think about the way the architecture will use the knowledge. What is not yet clear is why this same impasse won't arise the next time our model faces Sam Pro. Certainly, if Joe were a seasoned veteran who had faced Sam many times in his career we would expect his choice of pitch to be automatic. But won't our model of Joe always have to plod along, recalling past events and reasoning its way to the same conclusion? Won't the model always reach this impasse?

8. Learning

There is a very old joke in which a man carrying a violin case stops a woman on the street and asks, "How do you get to Carnegie Hall?" and the woman replies, "Practice." Her advice reflects one of the most interesting regularities in human behavior; when we do something repeatedly we generally get better at it. Better may mean that we make fewer errors, or do the task more quickly, or simply seem to require less effort to achieve some given level of performance. Somehow the act of doing the task *now* changes our ability to do the task in the future. In short, doing results in learning — the acquisition of knowledge through experience. It is this acquired knowledge that changes our behavior over time.

Two of the fundamental questions that arise when you talk about systems that learn are: What do they learn? And when do they learn? We can answer the first of these questions from what we know about Soar already: Soar systems learn new associations. To see why this must be so, remember our equation:

BEHAVIOR = ARCHITECTURE x CONTENT

Because learning leads to a change in behavior, either the architecture or the content must be affected by experience. Since the architecture is, by definition, a set of *fixed* structures and

mechanisms, it must be the content that changes. And since domain content is captured by associations, learning must lead to new associations.[6]

The answer to the question of when to learn is a bit harder to derive. Figure 8-1(a) shows part of the context that led to the impasse in Figure 7-1 in slightly different form and with the details of the state expanded. We call this the *pre-impasse environment*. When an impasse arises it means that the system does not have available in long-term memory associations for this context that lead to a single next move in the problem space. Expressed a bit more intuitively, an impasse is the architecture's way of signaling a lack of knowledge. A lack of knowledge is an opportunity for learning.

Now let's follow the processing in the resolve-tie subgoal, as shown in Figure 8-1(b). Association a9, given above, created Recall's initial state with nothing but information about the tied operators in it. The purpose of the Recall space is to model Joe deciding among the possible pitches by remembering how successful he's been with each in the past. Our model will accomplish this by augmenting the Recall state with cues from the pre-impasse environment until a relevant memory is evoked, i.e. until working memory contains elements that represent an event from Joe's past involving one or both of the tied pitches. Once such an event has been recalled, it can be used as the basis for evaluating each of the proposed pitches. There are many ways to produce this general behavior within the Soar architecture. The one we explore here requires that we add two kinds of knowledge to our model, knowledge that encodes memories of events in Joe's past, and knowledge that defines two new operators, augment and evaluate, that will evoke and evaluate those memories.

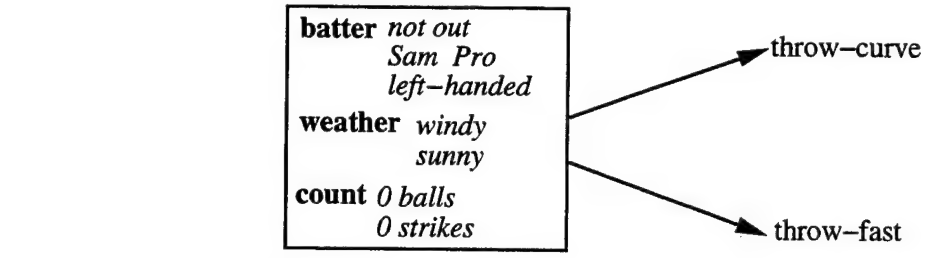
As in the case of modeling perception, modeling episodic memory is a complex topic that we will not address in detail. Instead, we will simply give Joe the following memory for demonstrative purposes:

(a10) If using the Recall problem space and the **batter** is *left-handed*
and the **weather** is *windy*
then add event116 to Recall's state with substructure
part-of *game22*, **isa** *homerun*, **pitch** *throw-fast*, etc.

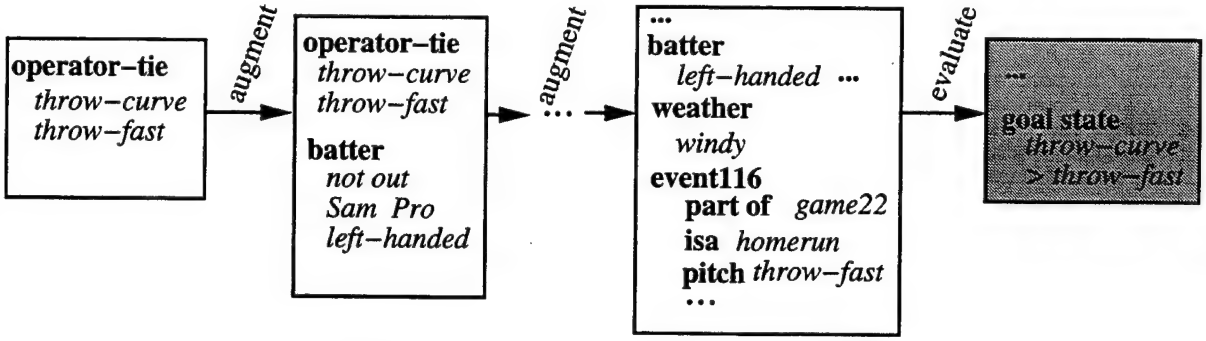
What is important to note about this association is that it does not fire in the Pitch problem space

---

[6]Strictly speaking, learning could lead to removing associations as well. However, Soar as a theory assumes that long-term memory only grows. So forgetting, in Soar, is a phenomenon that results when a new association prevents an old association from firing.



(a) Pre–impasse environment from the get–batter–out goal



(b) Processing in the resolve–tie subgoal

Figure 8-1: Learning when to prefer the curve ball. Elements in the higher goal context (the pre-impasse environment (a)) are used by operators available in the lower goal context (b) to elicit knowledge that resolves the operator tie. Here, the model elicits a ''memory'' of a previous game (event116) in which it was windy and throwing the fast ball to a left-handed batter led to a home run. On this basis, throwing the curve becomes preferred. The new association that will result from the resolution of this impasse will test for only those elements in the pre-impasse environment that led to the result, i.e. the windy day, left-handedness and the proposals of the two pitches.

although Pitch's state does have the necessary features in it. It seems unlikely that Joe is constantly being reminded about all the games in his past that share features with the current situation, and we don't want our model of Joe to be inundated with memories either. Instead, we want the model to be reminded when there is a reason for the reminding. In other words, the reminding is goal-driven; here, the goal that drives the reminding is the need to resolve the operator tie.

To elicit the memory of event116, Recall's state must contain the features and values in the ''if'' portion of association a10. To accomplish this, we use augment operators that copy a single feature and its values from Pitch's state to Recall's state:

(a11) If there is a feature in Pitch's state that is not in Recall's state
then suggest an augment operator for that feature

(a12) All augment operators are equally acceptable
(a13) If the augment operator has been selected in Recall for some feature
then add that feature and its values to Recall's state

Observe that association a11 proposes one augment operator for every feature that is in Pitch's state and not yet in Recall's state. Association a12, however, provides preference information about the proposed operators to the decision procedure. The preference information results in the random selection of a single instance of augment at the end of the decision cycle. The first state transition in Figure 8-1 shows the first application of an augment operator (via association a13) copying the batter information into Recall's state.

Remember that processing in the Recall problem space is driven by the goal of resolving the tie between the throw-curve and throw-fast-ball operators in Pitch. Since no useful knowledge has been elicited by the batter information alone, associations a11 through a13 fire again, selecting and copying a new feature and its values from Pitch. Eventually, however, an augment operator is selected that copies the **weather** feature. Once this has happened, the contents of working memory trigger association a10 (the episodic memory of event116), moving the model into the third state in Figure 8-1(b). In other words, the model has now been reminded of a previous game that has some elements in common with the current situation.

Once an episodic memory has been evoked, working memory contains information that can be used to try to resolve the tie. The next three associations define the evaluate operator. In the language of Table 3-1, this operator maps knowledge of particular episodes (K2) and knowledge of objectives (K4) into knowledge of when to choose actions or methods (K6):

(a14) If using the Recall problem space and there are two tied operators
and an **event** with a **pitch** that is one of the tied operators
then suggest an evaluate operator
(a15) Prefer evaluate operators over augment operators
(a16) If applying an evaluate operator to a state that has two tied operators
and an **event** that **isa** *homerun*
with a **pitch** that is one of the operators
then prefer the other operator

Association a14 proposes the evaluate operator on the basis of a relevant event. Association a15 is included to prefer an evaluation just in case there are other augment operators that are still possible. Association a16 adds a preference to working memory that results from applying the evaluate operator. In the current situation, a14-a16 simply mean that if Joe remembers a time when someone threw a fast ball that led to a homerun under similar circumstances, then he should prefer to throw the curve. As shown in the right-most state in Figure 8-1(b), the actual

working memory changes produced by association a16 include both a preference for the throw-curve operator and the delineation of the current state as a goal state in this problem space.

Look what just happened. The operator-tie impasse arose because the original goal context did not provide cues for eliciting the LTM knowledge to choose between two pitches. In response to the impasse, a subgoal to find the knowledge was created by the architecture. Domain content then expanded this subgoal into a second goal context involving the Recall problem space, i.e. the total context that could trigger relevant LTM knowledge (Pitch + Recall) was expanded. As a result of processing in Recall, the knowledge needed in Pitch has become available, and the impasse can be resolved. Moreover, we have an opportunity to learn a new association that will make that knowledge immediately available in Pitch under the circumstances that originally led to the impasse. In fact, the architecture automatically forms such a new association whenever results are generated from an impasse. To distinguish associations that are learned by the model from those created by the modeler, we give them a special name, *chunks*, and call the learning process that creates them *chunking*.

To create a chunk the architecture looks at every part of the context that existed in the pre-impasse environment (Figure 8-1(a)) that was used in reaching the result. In our example, the new association would be:

(c1) If using the Pitch problem space
and both the throw-curve and throw-fast-ball operators are suggested
and the **batter** is *left-handed* and the **weather** is *windy*
then prefer the throw-curve operator

As expected, the "if" portion of c1 includes only elements in the context before the impasse arose, and the "then" portion mirrors the knowledge elicited. With c1 in long-term memory, the model will not encounter an impasse the next time it is in a state like the one in Figure 8-1(a). Instead, this chunk will fire, the curve ball will be preferred, and processing will continue in Pitch.

Chunking is essentially a deductive, compositional learning mechanism; that is, the preference for a curve ball represents a kind of deduction from prior knowledge, and the new association is composed from a "then" part containing the deduction, and an "if" part containing the knowledge that contributed to the deduction. This recombination of existing knowledge makes it accessible in new contexts via new long-term memory associations. Notice that chunking overcomes some of the partitioning function of problem spaces (making knowledge available in

a problem space that it was not available in before learning), but that it does so only when experience dictates the need. As a theory, Soar says that chunking happens all the time — it is a ubiquitous mechanism that requires no intention to learn on the part of the agent. Impasses arise automatically when there is a lack of knowledge and chunks arise automatically when knowledge becomes available that helps resolve the impasse.

If chunking is appropriately characterized as deductive, can Soar systems still learn in all the ways humans do? Although chunking is the only *architectural* mechanism for learning, nothing about Soar keeps us from building other styles of learning as content theories that rest on top of the architecture. Indeed, since it is the only mechanism for changing the contents of long-term memory, chunking must be at the heart of any other learning method in Soar. Notice in our example that the augment operator is used to add features of the pre-impasse environment until a memory is elicited that has key features in common with the current situation. The system's behavior looks very much like reasoning by analogy, and the chunk that is learned may well be an overgeneralization (i.e., it may fire in situations where the preference it provides is actually inappropriate). Thus, even this simple model takes an inductive leap via the architecture's deductive mechanism. Other learning methods that have been demonstrated in Soar include learning by abduction (Johnson, 1994), and learning by instruction (Huffman, 1993).

9. Putting it all together: a Soar model of Joe Rookie

In our scenario, Joe Rookie chooses to throw a curve ball which is hit by the batter, caught by Joe on a bounce, and thrown to first base. Over the last five sections, we've moved from this narrative description of the activity of a rookie pitcher to a computational description of the architecture and content that produces that activity. Of course, we've only described a fraction of the knowledge involved in deciding to throw a curve ball and an even smaller fraction of the knowledge involved in playing out our original scenario. To expand the model to full functionality is beyond the scope of this chapter, but the method we would use should be clear. First, we specify the domain knowledge Joe needs; that's our content. Next, we tie the different types of domain knowledge to the different parts of the goal context: goals, problem spaces, state structures (including percepts and any other working memory elements that trigger actions), and operators. Finally, we specify the relationships between problem spaces by the impasses that will arise and the kinds of knowledge that will be missing, and consequently learned.

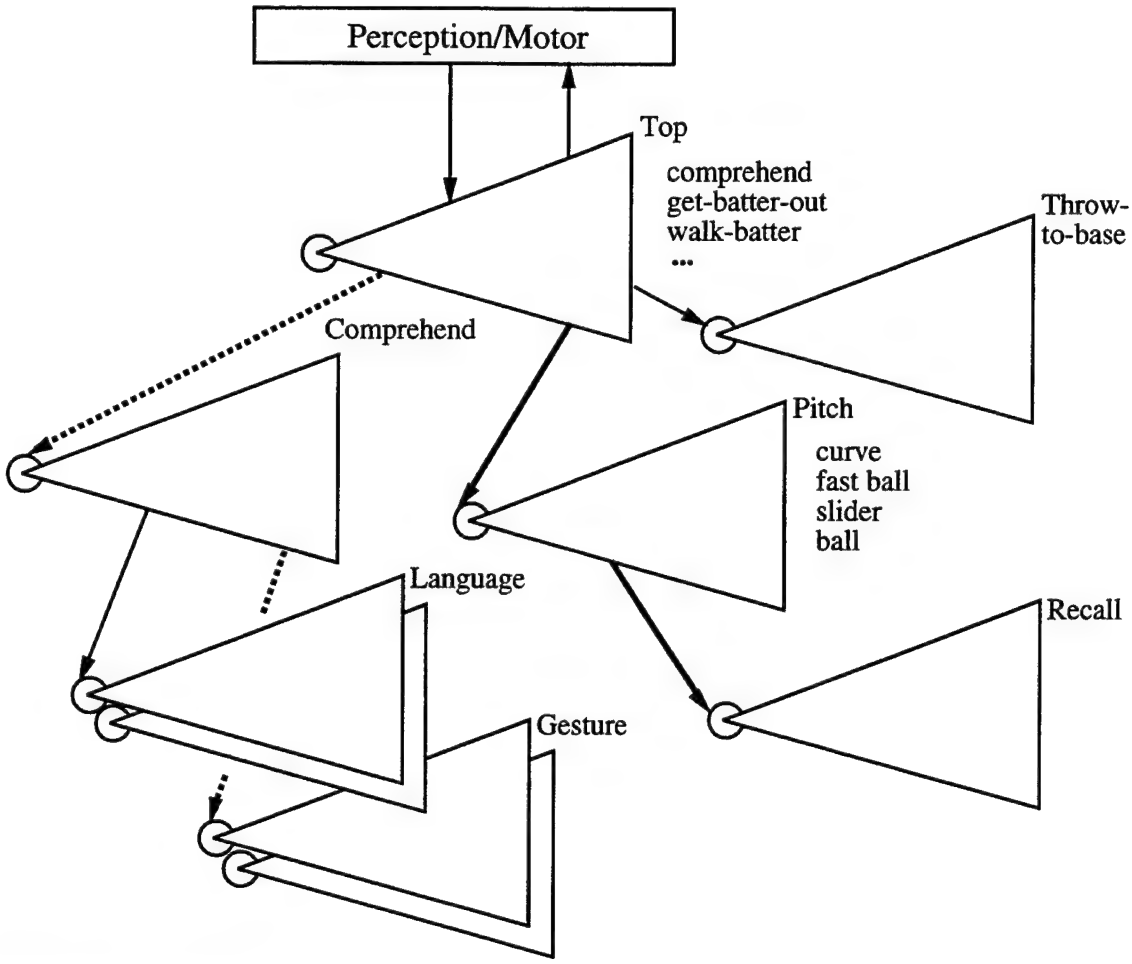


Figure 9-1: Expanding the model's capabilities. Our model of Joe needs far more knowledge to play out our original scenario than we have given it in the previous sections. Still, increasing the model's functionality can be done within the architectural framework we've described by adding more domain content in the form of goal contexts and LTM associations that map between them.

Figure 9-1 shows what a fuller model might look like in terms of problem spaces and their relationships. In Soar, the Top goal and Top problem space are given by the architecture and are the only goal and problem space that persist across time (other problem spaces and goals come and go as the goal hierarchy grows and shrinks). All perception and action occurs through the Top state so that they are part of the pre-impasse environment for all subgoals (this ensures that tests of percepts and actions will be preserved in any chunks that are learned). Operators in the Top problem space correspond to tasks Joe has to perform, like getting a batter out, comprehending language or gestures, and so on. Subgoal problem spaces implement these high-level tasks. Thus, our now-familiar Pitch problem space is actually evoked in response to an

impasse in Soar's Top problem space. Knowledge may be spread out among the subspaces in various ways, but will be integrated as a result of chunking in response to impasses. The dark arrows connect contexts in the goal hierarchy that arise while coming to the decision to throw the curve ball as we discussed it above. The chunk that arises in the Top space under this configuration would look like c1 except that it would check for the Top problem space and the task operator of getting the batter out (because these are elements in the pre-impasse environment that are used in arriving at the result).

Another possibility evident from the figure is that while Joe is in the Pitch problem space trying to decide on a pitch, the catcher might just signal the appropriate pitch. Under the configuration shown, the impasse from the Top space to Pitch would have to be terminated (by associations that prefer comprehension of the catcher's signal to getting the batter out), and the comprehend operator selected. Then an impasse on the comprehend operator would redirect processing along the dotted lines, into the Comprehend space and downward, in this case, to the Gesture problem space. The chunk resulting from the impasse on comprehend might map directly to the choice of a pitch and a motor action in the Top state. Alternatively, it might simply result in an augmentation to the Top state (e.g. with **suggested action** *throw-curve*). In this latter case, the get-batter-out operator would be reselected and impasse again, allowing the catcher's advice to be weighed against evidence from Recall in the subspace.

We could go on exploring alternatives to the model shown in Figure 9-1, elaborating it with more problem spaces and operators, moving operators or the connections between problem spaces around to explore the effect on learning, and so on. Clearly, the architecture permits many different ways of organizing domain content. It is possible that some of this variation must be there in order to model the natural variations in behavior that occur both within and across people. However, if even after taking such variability into account, the architecture still does not sufficiently constrain the structuring of the domain content, how has its specification helped us in our search for a unified theory of cognition? Before we address this question, let's step back and draw together in one place all the mechanisms and structures of the architecture we have developed above.

10. Stepping back: the Soar architecture in review

A cognitive architecture is really two things at once. First, it is a fixed set of mechanisms and structures that process content to produce behavior. At the same time, however, it is a theory, or point of view, about what cognitive behaviors have in common. In this chapter we have looked at one particular architecture, Soar, and so at one particular set of mechanisms and one particular point of view. Here is a brief synopsis of the mechanisms and structures we have discussed, along with some of the theoretical commitments that underlie them:

- **The Goal Context** is the structure at the heart of the architecture. Every goal context is defined by four slots and their values: the goal (ensures goal-directed behavior), the problem space (partitions or organizes knowledge in a goal-related way), the state (an internal representation of the situation), and the operator (maps from state to state).
- **Working Memory (WM)** contains the current situation (including past states and hypothetical states if they are important for reasoning) in the form of one or more goal contexts. WM also holds the results of perception as features and values in the Top state. The contents of working memory trigger both associations in LTM and motor actions.
- **Long-term Memory (LTM)** is the repository for domain content that is processed by the architecture to produce behavior. The form of knowledge in LTM is associations that map from the current goal context in WM to a new goal context. Because the mapping is from context to context, the triggering of an association can be accomplished by a simple match process against working memory. In addition to being associational, Soar's long-term memory is *impenetrable*. This means that a Soar system cannot examine its own associations directly; its only window into LTM is through the changes to working memory that are the results of association firings.
- **The Perception/Motor Interface** is the mechanism for defining mappings from the external world to the internal representation in working memory, and from the internal representation back out to action in the external world. Through this interface, perception and action can occur asynchronously and in parallel with cognition.
- **The Decision Cycle** is the basic architectural process supporting cognition. It is composed of two phases. Parallel access to LTM during the elaboration phase changes the features and values that define the state and suggests new values for context slots. Quiescence defines the moment at the end of the elaboration phase and the beginning of the decision phase when all the knowledge that can be elicited by the current context has been. During the second phase, the decision procedure interprets the domain-independent language of preferences and suggestions for changes to the context. The result of the decision procedure is either a single change to the goal context, or an impasse if no single change has been suggested. The application of a single operator per decision cycle imposes a *cognitive bottleneck* in the architecture, that is, a limit on how much work cognition can do at once.
- **Impasses** signal a lack of knowledge, and therefore, an opportunity for learning. An impasse occurs automatically whenever the knowledge elicited by the current context isn't enough for the decision procedure to resolve the preferences in working memory to a single change in the context. The language of impasses, like the language of preferences, is defined independently of any domain. When an impasse arises, the architecture also automatically begins the creation of a new subgoal context whose goal is to resolve the

impasse. In this way, impasses impose a goal/subgoal hierarchy on the contexts in working memory.

- **Chunking** is the pervasive architectural learning mechanism. Chunking automatically creates new associations in LTM whenever results are generated from an impasse. The new associations map the relevant pre-impasse WM elements into WM changes that prevent that impasse in future, similar situations. Although basically a deductive or compositional mechanism, chunking serves many purposes. It can integrate different types of knowledge that have been spread over multiple problem spaces. It can speed up behavior by compiling many steps through many subspaces into a single step in the pre-impasse problem space. It can be used as the basis of inductive learning, analogical reasoning, and so on. Because it is the only architectural mechanism for changing LTM, it is assumed to be the basis of all types of learning in people.

Using these structures and mechanisms we have shown how to model a small piece of cognitive behavior in the Soar architecture. Our purpose in building the model was to ground our discussion in some concrete, familiar behavior so it would be easy to see what was architectural — and, therefore, fixed — and what was not. It is important to realize, however, that the particular examples we modeled are only that, examples. To see how the Soar architecture can be used to help produce some other specific behavior, we would begin by viewing that behavior through the framework introduced in Section 3, describing the relevant goals that drive the behavior and the knowledge required to accomplish them. Once we have a knowledge-level description, we would tie the knowledge to the particular constructs in the goal context: to problem spaces, states, and operators. In partitioning the knowledge among problem spaces we would attend to the relationships defined by the different types of impasses so that learning results in meaningful new associations. These are the steps to building a Soar model, whether it is a model of a baseball pitcher, a student solving physics problems, or a jet fighter pilot engaging an enemy plane.

We have asked and answered a lot of questions about Soar. What structures and mechanisms are fixed? What structures and mechanisms are left up to the modeler? What processes occur automatically? What processes must be chosen deliberately? By answering these questions we have told you about the architecture we use. By asking them, we have tried to give you a method for analyzing and understanding other architectures.

The power behind the idea of cognitive architecture, any cognitive architecture, is that it lets us explore whether a single set of assumptions, a single theory of what is common among many cognitive behaviors, can in fact support all of cognition. However, defining the architecture, the

fixed mechanisms and structures, is only a step toward producing a unified theory of cognition. The architecture, by itself, does not answer all the questions we might want to ask about cognition. Still, given the architecture as a starting point, we know that additional theory must be cast as content that can be processed by that architecture. In short, the architecture is the framework into which the *content theories* must fit. Moreover, the framework provides the opportunity to combine content theories and watch how they interact — to move them from isolation into a unified system.

11. From architecture to unified theories of cognition

Our model of Joe Rookie is a content theory, although it is hard to imagine what in its content we might want to argue is general across human cognition. As a community of more than 100 researchers world-wide, the Soar project has contributed a number of more interesting content theories to the field, including:

- NL-Soar, a theory of human natural language (NL) comprehension and generation (Lehman, Lewis, & Newell, 1991; Rubinoff & Lehman, 1994) that has been used to explain many results from psycholinguistics, including the garden path phenomenon discussed in Section 1 (Lewis, 1993).
- SCA, a theory of symbolic concept learning that acquires and retrieves category prediction rules (e.g. if you see something round and small that rolls, predict it is a ball) (Miller, 1993). SCA mirrors psychological data along the dimensions of response time, accuracy and learning rate as a function of various instance and category structures.
- NOVA, a theory of visual attention whose performance matches human data from a number of distinct experimental results in the psychology literature (Wiesmeyer, 1992). Using a single set of mechanisms, NOVA explains regularities in human reaction times under various precuing and stimulus conditions, as well as regularities in the loss of memory for visual stimuli. Although we can view NOVA as a content theory with respect to the current Soar implementation, most of its content constitutes a theory of how the cognitive and vision systems interact. In this sense, we can also view NOVA as an architectural extension.

Each of these content theories has, in its own way, been an attempt at unification because each has explained a set of phenomena that had not been explained by a single theory previously. Moreover, because these content theories have been expressed computationally as Soar models, they all share the assumptions in the Soar architecture. What this means is that even though a content theory models aspects of human language (or concept learning, or vision), it does so within a framework that makes certain assumptions about human problem solving, memory, etc. The resulting model may not, itself, show the full range of human problem solving or memory phenomena, but the theory will, at the very least, be compatible with what is assumed to be

architectural in the other content theories. In this weak sense, content theories like the ones above constitute a burgeoning unified theory of cognition (UTC).

Of course, the stronger notion of Soar as a UTC requires that these individual content theories all work together in a single computational model. A number of other Soar projects have focused on variations of that goal, producing models such as:

- NTD-Soar, a computational theory of the perceptual, cognitive, and motor actions performed by the NASA Test Director (NTD) as he utilizes the materials in his surroundings and communicates with others on the Space Shuttle launch team (Nelson, Lehman, & John, 1994). NTD-Soar combined NL-Soar and NOVA with decision making and problem solving knowledge relevant to the testing and preparation of the Space Shuttle before it is launched.
- Instructo-Soar, a computational theory of how people learn through interactive instruction (Huffman, 1993). Instructo-Soar combined NL-Soar with knowledge about how to use and learn from instructions during problem solving to produce a system that learned new procedures through natural language.
- IMPROV, a computational theory of how to correct knowledge about what actions do in the world (Pearson & Laird, 1995). IMPROV combined SCA with knowledge about how to detect when the system's internal knowledge conflicts with what it observes in the world so that the system can automatically improve its knowledge about its interactions with the world.

The complex behavior achieved by each of these systems was made possible, in part, because some component content theories were already available. Using Soar as a common framework meant that there already existed a language capability (or visual attention mechanism, or concept acquisition mechanism) cast in the right terms and ready (more or less) for inclusion. From the point of view of the NTD project, for example, the availability of NL-Soar meant that they didn't have to build a theory of human language in addition to a theory of the launch domain. More importantly, incorporating NL-Soar meant that the whole model had to "play by the rules" linguistically. NTD-Soar couldn't be constructed to wait to pay attention to linguistic input indefinitely because NL-Soar says that acoustic information doesn't stay around indefinitely. It couldn't wait until the end of the sentence to produce the meaning because NL-Soar says meaning has to be produced incrementally, word-by-word. In essence, NTD-Soar was constrained by incorporating NL-Soar. But, of course, the constraints hold in both directions. From the point of view of the NL-Soar project, the NASA Test Director's linguistic environment presented many challenges that had not originally shaped the model. Indeed, NL-Soar had been created as a model of comprehension and generation in isolation. The need to actually comprehend and generate sentences while doing other things changed the structure of NL-Soar

for all time.

As these examples show, we are still a long way from creating a full unified theory of cognition. As a candidate UTC, Soar currently insists that theories of regularities in behavior be cast in terms of associational memory, goal contexts, impasses, and chunking. Whether these are the right assumptions, whether we can construct on top of these assumptions a set of computationally-realizable mechanisms and structures that can answer all the questions we might want to ask about cognitive behavior, is an empirical question. Still, our methodology in working toward that goal is clear: work within the architecture, base content theories on the regularities already available from the contributing disciplines of cognitive science, and combine those content theories to try to explain increasingly complex behaviors. In short, keep searching for the big picture.

12. References

Anderson, J. R. 1993. *Rules of the Mind.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Fitts, P. M. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology, 47*, 381-391.

Gibson, E. 1990. Recency preference and garden-path effects. *Twelfth Annual Conference of the Cognitive Science Society*, 372-379 .

Huffman, S. B. 1993. *Instructable Autonomous Agents.* Ph.D. diss., The University of Michigan. Department of Electrical Engineering and Computer Science.

Johnson, T. R., Krems, J. & Amra, N. K. 1994. A computational model of human abductive skill and its acquisition. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 463-468 .

Kieras, D.E. & Meyer, D.E. 1994. *The EPIC architecture for modeling human information-processing: A brief introduction.* Technical Report , Ann Arbor, University of Michigan, Department of Electrical Engineering and Computer Science, TR-94/ONR-EPIC-1.

Lehman, J. Fain, Lewis, R., & Newell, A. 1991. Integrating knowledge sources in language comprehension. *Proceedings of the Thirteenth Annual Conferences of the Cognitive Science Society*, 461-466 .

Lewis, R. L. 1993. *An Architecturally-based Theory of Human Sentence Comprehension.* Ph.D. diss., Carnegie Mellon University. Also available as Technical Report CMU-CS-93-226.

Miller, C. S. 1993. *Modeling Concept Acquisition in the Context of a Unified Theory of Cognition.* Ph.D. diss., The University of Michigan. Also available as Technical Report CSE-TR-157-93.

Nelson, G., Lehman, J. F., John, B. 1994. Integrating cognitive capabilities in a real-time task. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 658-663 .

Newell, A. 1990. *Unified Theories of Cognition.* Cambridge, Massachusetts: Harvard University Press.

Newell, A. & Simon, H. 1972. *Human Problem Solving*. Englewood Cliffs, New Jersey: Prentice-Hall.

Nilsson, Nils. 1971. *Problem-solving Methods in Artificial Intelligence*. New York, New York: McGraw-Hill.

Pearson, D. J. & Laird, J. E. 1995. Toward Incremental Knowledge Correction for Agents in Complex Environments. In S. Muggleton, D. Michie, and K. Furukawa (Ed.), *Machine Intelligence*. Oxford University Press. forthcoming.

Pritchett, B. L. 1988. Garden path phenomena and the grammatical basis of language processing. *Language 64*, 539-576.

Rubinoff, R. & Lehman, J. F. 1994. Real-time natural language generation in NL-Soar. *Proceedings of the Seventh International Workshop on Natural Language Generation*, 199-206 .

Sternberg, S. 1975. Memory scanning: New findings and current controversies. *Quarterly Journal of Experimental Psychology, 27*, 1-32.

The Commissioner of Baseball. 1994. *Official Baseball Rules, 1994 Edition*. St. Louis, MO: Sporting News Publishing Co.

Tulving, E. 1983. *Elements of Episodic Memory*. New York, New York: Oxford University Press.

Wiesmeyer, M. D. 1992. *An Operator-Based Model of Human Covert Visual Attention*. Ph.D. diss., University of Michigan.